Docling Technical Report

Version 1.0

Christoph Auer Maksym Lysak Ahmed Nassar Michele Dolfi Nikolaos Livathinos
Panos Vagenas Cesar Berrospi Ramis Matteo Omenetti Fabian Lindlbauer
Kasper Dinkla Lokesh Mishra Yusik Kim Shubham Gupta Rafael Teixeira de Lima
Valery Weber Lucas Morin Ingmar Meijer Viktor Kuropiatnyk Peter W. J. Staar

AI4K Group, IBM Research
Rüschlikon, Switzerland

Abstract

This technical report introduces *Docling*, an easy to use, self-contained, MIT-licensed open-source package for PDF document conversion. It is powered by state-of-the-art specialized AI models for layout analysis (DocLayNet) and table structure recognition (TableFormer), and runs efficiently on commodity hardware in a small resource budget. The code interface allows for easy extensibility and addition of new features and models.

1 Introduction

Converting PDF documents back into a machine-processable format has been a major challenge for decades due to their huge variability in formats, weak standardization and printing-optimized characteristic, which discards most structural features and metadata. With the advent of LLMs and popular application patterns such as retrieval-augmented generation (RAG), leveraging the rich content embedded in PDFs has become ever more relevant. In the past decade, several powerful document understanding solutions have emerged on the market, most of which are commercial software, cloud offerings [3] and most recently, multi-modal vision-language models. As of today, only a handful of open-source tools cover PDF conversion, leaving a significant feature and quality gap to proprietary solutions.

With *Docling*, we open-source a very capable and efficient document conversion tool which builds on the powerful, specialized AI models and datasets for layout analysis and table structure recognition we developed and presented in the recent past [12, 13, 9]. Docling is designed as a simple, self-contained python library with permissive license, running entirely locally on commodity hardware. Its code architecture allows for easy extensibility and addition of new features and models.

Here is what Docling delivers today:

- Converts PDF documents to JSON or Markdown format, stable and lightning fast
- Understands detailed page layout, reading order, locates figures and recovers table structures
- Extracts metadata from the document, such as title, authors, references and language
- Optionally applies OCR, e.g. for scanned PDFs
- Can be configured to be optimal for batch-mode (i.e high throughput, low time-to-solution) or interactive mode (compromise on efficiency, low time-to-solution)
- Can leverage different accelerators (GPU, MPS, etc).

2 Getting Started

To use Docling, you can simply install the *docling* package from PyPI. Documentation and examples are available in our GitHub repository at github.com/DS4SD/docling. All required model assets[1] are downloaded to a local huggingface datasets cache on first use, unless you choose to pre-install the model assets in advance.

Docling provides an easy code interface to convert PDF documents from file system, URLs or binary streams, and retrieve the output in either JSON or Markdown format. For convenience, separate methods are offered to convert single documents or batches of documents. A basic usage example is illustrated below. Further examples are available in the Doclign code repository.

```
from docling.document_converter import DocumentConverter

source = "https://arxiv.org/pdf/2206.01062"  # PDF path or URL
converter = DocumentConverter()
result = converter.convert_single(source)
print(result.render_as_markdown())  # output: "## DocLayNet: A Large
    Human-Annotated Dataset for Document-Layout Analysis [...]"
```

Optionally, you can configure custom pipeline features and runtime options, such as turning on or off features (e.g. OCR, table structure recognition), enforcing limits on the input document size, and defining the budget of CPU threads. Advanced usage examples and options are documented in the README file. Docling also provides a *Dockerfile* to demonstrate how to install and run it inside a container.

3 Processing pipeline

Docling implements a linear pipeline of operations, which execute sequentially on each given document (see Fig. 1). Each document is first parsed by a PDF backend, which retrieves the programmatic text tokens, consisting of string content and its coordinates on the page, and also renders a bitmap image of each page to support downstream operations. Then, the standard model pipeline applies a sequence of AI models independently on every page in the document to extract features and content, such as layout and table structures. Finally, the results from all pages are aggregated and passed through a post-processing stage, which augments metadata, detects the document language, infers reading-order and eventually assembles a typed document object which can be serialized to JSON or Markdown.

3.1 PDF backends

Two basic requirements to process PDF documents in our pipeline are a) to retrieve all text content and their geometric coordinates on each page and b) to render the visual representation of each page as it would appear in a PDF viewer. Both these requirements are encapsulated in Docling's PDF backend interface. While there are several open-source PDF parsing libraries available for python, we faced major obstacles with all of them for different reasons, among which were restrictive

[1] see huggingface.co/ds4sd/docling-models/

torch runtimes backing the Docling pipeline. We will deliver updates on this topic at in a future version of this report.

Table 1: Runtime characteristics of Docling with the standard model pipeline and settings, on our test dataset of 225 pages, on two different systems. OCR is disabled. We show the time-to-solution (TTS), computed throughput in pages per second, and the peak memory used (resident set size) for both the Docling-native PDF backend and for the pypdfium backend, using 4 and 16 threads.

CPU	Thread budget	native backend			pypdfium backend		
		TTS	Pages/s	Mem	TTS	Pages/s	Mem
Apple M3 Max (16 cores)	4	177 s	1.27	6.20 GB	103 s	2.18	2.56 GB
	16	167 s	1.34		92 s	2.45	
Intel(R) Xeon E5-2690 (16 cores)	4	375 s	0.60	6.16 GB	239 s	0.94	2.42 GB
	16	244 s	0.92		143 s	1.57	

5 Applications

Thanks to the high-quality, richly structured document conversion achieved by Docling, its output qualifies for numerous downstream applications. For example, Docling can provide a base for detailed enterprise document search, passage retrieval or classification use-cases, or support knowledge extraction pipelines, allowing specific treatment of different structures in the document, such as tables, figures, section structure or references. For popular generative AI application patterns, such as retrieval-augmented generation (RAG), we provide *quackling*, an open-source package which capitalizes on Docling's feature-rich document output to enable document-native optimized vector embedding and chunking. It plugs in seamlessly with LLM frameworks such as LlamaIndex [8]. Since Docling is fast, stable and cheap to run, it also makes for an excellent choice to build document-derived datasets. With its powerful table structure recognition, it provides significant benefit to automated knowledge-base construction [11, 10]. Docling is also integrated within the open IBM data prep kit [6], which implements scalable data transforms to build large-scale multi-modal training datasets.

6 Future work and contributions

Docling is designed to allow easy extension of the model library and pipelines. In the future, we plan to extend Docling with several more models, such as a figure-classifier model, an equation-recognition model, a code-recognition model and more. This will help improve the quality of conversion for specific types of content, as well as augment extracted document metadata with additional information. Further investment into testing and optimizing GPU acceleration as well as improving the Docling-native PDF backend are on our roadmap, too.

We encourage everyone to propose or implement additional features and models, and will gladly take your inputs and contributions under review. The codebase of Docling is open for use and contribution, under the MIT license agreement and in alignment with our contributing guidelines included in the Docling repository. If you use Docling in your projects, please consider citing this technical report.

References

[1] J. AI. Easyocr: Ready-to-use ocr with 80+ supported languages. https://github.com/JaidedAI/EasyOCR, 2024. Version: 1.7.0.

[2] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala. Pytorch 2: Faster